Data Visualization

Data visualization is a very important aspect of analysing climate data. Visualization as a part of analysis and creating presentation quality graphics are accomplished in CDAT by using the point and click GUI in VCDAT, from the CDAT command line, or in a program. The primary tool inside CDAT to visualize the data is the Visualization and Control System (VCS). Additionally, because of the ease of controlling other applications using Python as a link language, an interface to Grace (an open source application) is also available to users.

Visualization and Control System (VCS)

VCS is expressly designed to meet the needs of climate scientists. Because of the breadth of its capabilities, VCS can be a useful tool for other scientific applications as well. VCS allows wide−ranging changes to be made to the data display, provides for presentation hardcopy output, and includes a means for recovery of a previous display.

In the VCS model, the data display is defined by a trio of named object sets, designated the" primary objects" (or "primary elements"). These include:

- the data, which define what is to be displayed and is ingested via other CDAT software components;
- the graphics method, which specifies the display technique; and
- the picture template, which determines the appearance of each segment of the display.

Tables for manipulating these primary objects are stored in VCS for later recall and possible use. In addition, detailed specification of the primary objects' attributes is provided by eight "secondary objects" (or "secondary elements"):

- colormap: specification of combinations of 256 available colors
- fillarea: style, style index, and color index
- format: specifications for converting numbers to display strings
- line: line type, width and color index
- list: a sequence of pairs of numerical and character values
- marker: marker type, size, and color index
- texttable: text font type, character spacing, expansion and color index
- textorientation: character height, angle, path, and horizontal/vertical alignment

By combining primary and secondary objects in various ways (either at the command line or in a program), the VCS user can comprehensively diagnose and inter−compare climate model simulations. VCS provides capabilities to:

- Create and modify existing template attributes and graphics methods
- Save the state−of−the−system as a script to be run interactively or in a program
- Save a display as a Computer Graphics Metafile (CGM), GIF, Postscript, Sun Raster, or Encapsulated Postscript file
- Create and modify colormaps
- Zoom into a specified portion of a display
- Change the orientation (portrait vs. landscape) or size (partial vs. full−screen) of a display
- Animate a single data variable or more than one data variable simultaneously
- Display different map projections

Displaying Data

VCS can handle the CDMS data objects (such as Transient Variables seen earlier in addition to Numeric arrays, MA arrays, python lists and tuples). Plotting data is simply accomplished by importing the vcs module and issuing the plot command. For example:

```
import cdms, vcs
f =cdms.open('example.nc')
my_data = f('clt')
v = vcs.init()
v.plot(my_data)
```

The plot will display the data "my_data" using default settings. However, the user can control every aspect of the plot's appearance individually. The first of those is the "Graphics Method" described in the next section.

Graphics Methods

A graphics method simply defines how data is to be displayed on the screen. Currently, there are eleven different graphics methods with more on the way. Each graphics method has its own unique set of attributes (or members) and functions. They also have a set of core attributes that are common in all graphics methods. The descriptions of the current set of graphics methods are as follows:

- **boxfill** – The boxfill graphics method draws color grid cells to represent the data on the VCS Canvas.
- **isofill** – The isofill graphics method fills the area between selected isolevels (levels of constant value) of a two–dimensional array with a user–specified color.
- **isoline** – The isoline graphics method draws lines of constant value at specified levels in order to graphically represent a two–dimensional array. It also labels the values of these isolines on the VCS Canvas.
- **outfill** – The outfill graphics method fills a set of integer values in any data array. Its primary purpose is to display continents by filling their area as defined by a surface type array that indicates land, ocean, and sea–ice points.
- **outline** – The Outline graphics method outlines a set of integer values in any data array. Its primary purpose is to display continental outlines as defined by a surface type array that indicates land, ocean, and sea–ice points.
- **scatter** – The scatter graphics method displays a scatter plot of two 4–dimensional data arrays, e.g. A(x,y,z,t) and B(x,y,z,t).
- **vector** – The Vector graphics method displays a vector plot of a 2D vector field. Vectors are located at the coordinate locations and point in the direction of the data vector field. Vector magnitudes are the product of data vector field lengths and a scaling factor.
- **xvsy** – The XvsY graphics method displays a line plot from two 1D data arrays, that is X(t) and Y(t), where 't' represents the 1D coordinate values.
- **xyvsy** – The Xyvsy graphics method displays a line plot from a 1D data array, i.e. a plot of X(y) where 'y' represents the 1D coordinate values.
- **yxvsx** – The Yxvsx graphics method displays a line plot from a 1D data array, i.e. a plot of Y(x) where 'x' represents the 1D coordinate values.

Say for example you would like to plot the data object "my_object" using the "isofill" graphics method (instead of the default "boxfill" method), you would type:

```
v.isofill(my_data)
```

The user can control any aspect of the isofill method to get the precise appearance on the plot. To alter the isofill methods for use in your plot it will be necessary to "create" an isofill object. The create functions allow the user to create VCS objects which can be modified directly to produce the desired results. Since the VCS "default" objects allow for modifications, it is best to either create a new VCS object or get an existing one. When a VCS object is created, it is stored in an internal table for later use and/or recall.

```
my_new_isofill = v.createisofill('newisofillname')
```

To show the list of existing isofill graphics methods type:

```
v.show('isofill')
```

If there is an existing isofill method you have created and would like to use it (or alter it), then type:

```
my_old_isofill = v.getisofill('existing_isofillobjectname')
```

or use this "existing_isofillname" as a base to create the new one:

```
new_isofill2 = v.createisofill('newisofillname', 'existing_isofillobjectname')
```

The list of attributes can be viewed with the following command:

```
my_old_isofill.list()
```

You can explicitly set each of the attributes. For example, to change the levels to be used in plotting:

```
my_old_isofill.levels = [2., 3., 4., 6]
```

The graphics method can also be used as an optional argument to the plot command so that VCS now allows you to issue the command:

```
v.plot(my_data, my_old_isofill)

# You can also pass the name of the graphics
# method object or the object itself as shown below
v.plot(my_data, "isofill", "isofill_name")
```

To enable the user to check whether an object is of a certain graphics method (or any VCS object), there are a whole set of query functions made available. Therefore you can check if my_old_isofill is an isofill object and get a yes/no (1|0) answer by saying:

```
my_old_isofill.isisofill()
```

For help with queries type (at the shell prompt):

```
% pydoc vcs.queries
```

For more information you can type:

```
vcs.help('plot')
# or
print v.plot.__doc__
```

Extensive documentation can be found in Visualization and Control System: Command Line and Application Programming Interface (vcs.pdf).

Graphics Primitives

Graphics primitives are useful in enhancing plots and in creating additional ways of displaying data beyond the existing graphics methods. These primitive objects are created and altered using the same syntax as in the case of graphics methods. The following primitive objects are available:

- **fillareaobject** – The fillarea objects allows the user to edit fillarea attributes, including fillarea interior style, style index, and color index. The fill area attributes are used to display regions defined by closed polygons, which can be filled with a uniform color, a pattern, or a hatch style. Attributes specify the style, color, position, and dimensions of the fill area.
- **lineobject** – The line object allows the editing of line type, width, and color index. The line attributes specify the type, width, and color of the line to be drawn for a graphical display.
- **markerobject** – The marker object allows the editing of the marker type, width, and color index. The marker attribute specifies graphical symbols, symbol sizes, and colors used in appropriate graphics methods.
- **textobject** – Graphical displays often contain textual inscriptions, which provide further information. The text–table object attributes allow the generation of character strings on the VCS Canvas by defining the character font, precision, expansion, spacing, and color. The text–orientation object attributes allow the appearance of text character strings to be changed by defining the character height, up–angle, path, and horizon tal and vertical alignment. The text–combined object is a combination of both text–table and text–orientation objects.

Templates

Picture Template attributes describe where and how segments of a picture will be displayed. The segments are graphical representations of: textual identification of the data formatted values of single–valued dimensions and mean, maximum, and minimum data values axes, tick marks, labels, boxes, lines, and a legend that is graphics–method specific to the data. Picture templates describe where to display all segments including the data.

Templates also can be created and altered just like the graphics methods seen in the previous section. The syntax is retained the same for all objects in VCS so as to avoid confusion. The general philosophy behind templates is – "You should be able to specify the behaviour of every picture segment – text, data, line, etc. precisely according to your needs."

By setting the "priority" attribute of each picture segment you can control the order in which segments are drawn and whether they are drawn at all. The higher the priority number (integer value), the later it is drawn during plot creation ensuring that it is on top. Setting priority = 0 means you do not want it drawn!

The following segments of a template can be controlled and the values that can be set are:

- **data** – The location where the data is plotted (x1, x2, y1, y2) and its priority can be specified.
- **title** – The location(x, y), priority and the text objects (font type, size, angle, justification, etc. etc.) (More on text objects later). The title plotted is the title of your data read in. If you are plotting the variable "tdata", tdata.title is shown in the location specified for title.

- **units** – The data units. For example: "Degrees C" You can set the x,y location, priority and text object (more later on text objects).
- **dataname** – The name of the variable.
- **source** – The data source description.
- **function** – If computed data, the user can give the algebraic equation.
- **file** – The location of the file.
- **crdate** – The date of creation of plot. You can control x,y location, priority, and text object.
- **crtime** – The time of creation of plot. You can control x,y location, priority, and text object.
- **mean**, **max**, **min** – The values are computed automatically from the data you are plotting and you can set the x,y location, text object, priority and display format.
- **legend** – You can set the legend bar location and dimensions (x1, x2, y1, y2), the line type object, text object and of course priority.
- **comment1**, **comment2**, **comment3**, **comment4** – Four text comments can be drawn. For each of these you can set priority, location(x, y), and text object.You would have to set the comment on the data. Say you are plotting "tdata" using the x.plot(tdata) command, then you need to have done:

- **line1**, **line2**, **line3**, **line4** – Four lines can be drawn. For each of these lines you can set priority, start location (x1, y1), end location (x2, y2) and line object.
- **box1**, **box2**, **box3**, **box4** – Four boxes can be drawn. Same as line except the x1, x2, y1, y2 settings refer to corners of the box.
- **xname**, **yname**, **zname**, **tname** – These are the possible axes of x,y,z and t. Note in the case you are trying the plot is a latitude x longitude plot. If it was a timexlongitude plot then you would be setting the xname and tname values. You can set the x,y location for the name, the priority and text object.
- **xunits**, **yunits**, **zunits**, **tunits** – These are the respective axis units for whichyou can set the x,y location, text object, and priority .
- **xvalue**, **yvalue**, **zvalue**, **tvalue** –
- **xlabel1**, **xlabel2** – The x axis labels (bottom and top of your plot) can be independently set. You can specify the y location (x is determined by the data), priority and the text object.
- **ylabel1**, **ylabel2** – Same as above except for the left and right side of your plot. Here you can only specify the x location of the label.
- **xtic1**, **xtic2** – The major tic marks on the bottom and top. You can control the priority, y1 and y2 (in effect specifying the length!), and line object (Things like line style, thickness, arrow, etc. etc. More on the line object later).
- **xmintic1**, **xmintic2** – The minitic specifications. Exactly the same as above otherwise.
- **ytic1**, **ytic2** – The major tic marks on the left and right. You can control the priority, x1 and x2 (in effect specifying the length!), and line object.
- **ymintic1**, **ymintic2** – The minitic specifications. Exactly the same as above otherwise.

In addition to the attributes that can be specified, a few useful functions that operate on templates are also available.

- **scale** – This is useful in scaling the entire template (and all the objects within) by a specified factor.

- **move –** This is useful in moving the template (and all the objects within) to a specified location

Animation

VCS allows the user to animate the contents of the VCS Canvas. This function pops up the animation GUI which lets the user control all aspects of the animation.

```
v.plot(array,'default','isofill','quick')
v.animate.gui()
```

Alternately, animation can be controlled from the command line using the following methods:

```
v.animate.create()
v.animate.run()
v.animate.zoom(3)
v.animate.horizontal(50)
v.animate.vertical(-50)
v.animate.pause(4)
v.animate.frame(2)
v.animate.stop()
```

Output

Before attempting to print your plots, make sure that gplot is built and installed on your system. The VCS graphics can be output to files of various formats or directly to printers with postscript capability. The available graphics file formats that one can print to are postscript, encapsulated postscript (EPS), GIF, CGM, and raster. To print directly to a printer and optionally specifying a portrait orientation:

```
v.printer('printer_name', 'p')
```

To create gif, postscript, cgm, raster, encapsulated postscript and portable document format (pdf) files, the commands are:

```
v.gif('example.gif','r','p')
v.postscript('example.ps', 'r', 'p')
v.cgm('example.cgm', 'p')
v.raster('example.ras', 'p')
v.eps('example.eps', 'r', 'p')
v.pdf('example.pdf')
```

Using VCS Scripts

Script commands define the actions that are necessary to preserve an interactive session as a script and to mimic that session in a non–interactive replay of the script. Many attributes are needed to create a graphical representation of a variable, e.g. attributes to identify the variable and to label the plotting axes. By use of

VCS and Python scripts, most of these attributes can be manipulated to create the desired visual effect, and the resulting attributes can be saved for later use. VCS and Python scripts also allow the user to save a sequence of interactive operations for replay, and to recover from a system failure. To re−save the initial.attributes file, use the function saveinitialfile(). To save VCS objects as Python scripts or VCS scripts, use the function scriptobject(). To save the state of the system, use the function scriptstate(). To run a VCS script file, use the function scriptrun()

Interface to Grace (genutil.xmgrace)

Nothing emphases the fact that CDAT is a collection of tools that can be extended by the user better than the xmgrace module. This module provides an interface to the popular Grace plotting utility (which you must have installed separately. Downloads and information are available from http://plasma−gate.weizmann.ac.il/Grace). The xmgrace tutorial which is introduced in xmgrace_tutorial.py teaches you how to use it.